Name : Jadhav Laxman

Class : SYA

Roll No : 138

Subject : DAA

ASSIGNMENT - 02

1. Differentiate between Backtracking and Branch & Bound

Ans :

- Backtracking is an algorithm that solves the problem in a recursive manner. It is a systematic way of trying different sequences of decisions to find the correct decision
- Branch and bound is more suitable for situations where we cannot apply the greedy method and dynamic programming. Usually, this algorithm is slow as it requires exponential time complexities during the worst case, but sometimes it works with reasonable efficiency

The ***main difference*** between backtracking and branch and bound is that

- Backtracking is an algorithm for finding all solutions to some computational problems, notably constraint satisfaction problems that incrementally builds candidates to the solutions. Branch and bound is an algorithm for discrete and combinatorial optimization problems and mathematical optimization

- Moreover, **efficiency** is a major difference between backtracking and branch and bound. Backtracking is more efficient than the Branch and Bound algorithm.

BACKTRACKING	BRANCH AND BOUND
An algorithm for finding all solutions to some computational problems, notably constraint satisfaction problems that incrementally builds candidates to the solutions	An algorithm for discrete and combinatorial optimization problems and mathematical optimization
Finds the solution to the overall issue by finding a solution to the first subproblem and them recursively solving other subproblems based on the solution of the first issue	Solves a given problem by dividing it into at least two new restricted subproblems
More efficient	Less efficient

2. Why previously all P- type algorithms were NP-type algorithms? Justify.

OR

Why P-type algorithms is subset of NP-type algorithms? Justify.

Ans :

- Let's define P and NP.
  P is the set of all decision problems that you can solve in polynomial time, i.e. The running time of an algorithm that decides an instance of a particular problem *p* runs in time $O(n^k)$ for some choice of $k \in R$.
  NP is the set of all decision problems that when given an instance of a problem *p* and a candidate solution *c*, you can check to see if c really is the solution in polynomial time.

- Theorem: *P* is a subset of *NP*.

- Proof: Suppose problem p1 is in P and you want to show that p_1 is also in NP. Given a particular problem instance i of p1 and a candidate solution ci , the following algorithm will decide if ci is really the correct solution:
  1. Since p1p1is in P, we have a polynomial time algorithm that can be used to determine the answer for pipi. Run this algorithm on pipi and get a yes or no answer.
  2. Returns this as the answer.

- Another theory:

P is a subset of NP

- Let prob be a problem in P
- There is a deterministic algorithm that solves prob in polynomial time *O(nk),* for some constant *k*
- That same algorithm alg runs in a nondeterministic machine (it just do not use the oracle)

- Thus, alg has the same polynomial complexity, *O(nk)*
- Thus, prob is in NP.